Cardinality of Equivalence Relations

Lukas Bulwahn

February 23, 2021

Abstract

This entry provides formulae for counting the number of equivalence relations and partial equivalence relations over a finite carrier set with given cardinality.

To count the number of equivalence relations, we provide bijections between equivalence relations and set partitions [4], and then transfer the main results of the two AFP entries, Cardinality of Set Partitions [1] and Spivey’s Generalized Recurrence for Bell Numbers [2], to theorems on equivalence relations. To count the number of partial equivalence relations, we observe that counting partial equivalence relations over a set $A$ is equivalent to counting all equivalence relations over all subsets of the set $A$. From this observation and the results on equivalence relations, we show that the cardinality of partial equivalence relations over a finite set of cardinality $n$ is equal to the $n + 1$-th Bell number [3].

Contents

1 Cardinality of Equivalence Relations 2
  1.1 Bijection between Equivalence Relations and Set Partitions . 2
    1.1.1 Possibly Interesting Theorem for HOL.Equiv-Relations 2
    1.1.2 Dedicated Facts for Bijection Proof . . . . . . . . . . . 2
    1.1.3 Bijection Proof . . . . . . . . . . . . . . . . . . . . . 3
  1.2 Finiteness of Equivalence Relations . . . . . . . . . . . . . . . 4
  1.3 Cardinality of Equivalence Relations . . . . . . . . . . . . . . 4

2 Cardinality of Partial Equivalence Relations 5
  2.1 Definition of Partial Equivalence Relation . . . . . . . . . . . 5
  2.2 Construction of all Partial Equivalence Relations for a Given
      Set . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.3 Injectivity of the Set Construction . . . . . . . . . . . . . . . 7
  2.4 Cardinality Theorem of Partial Equivalence Relations . . . . 8
1 Cardinality of Equivalence Relations

theory Card-Equiv-Relations
imports
  Card-Partitions, Card-Partitions
  Bell-Numbers-Spivey.Bell-Numbers
begin

1.1 Bijection between Equivalence Relations and Set Partitions

1.1.1 Possibly Interesting Theorem for HOL.Equiv-Relations

This theorem was historically useful in this theory, but is now after some proof refactoring not needed here anymore. Possibly it is an interesting fact about equivalence relations, though.

lemma equiv-quotient-eq-quotient-on-UNIV:
  assumes equiv A R
  shows A // R = (UNIV // R) − {{}}
proof
  show UNIV // R − {{}} ⊆ A // R
  proof
    fix X
    assume X ∈ UNIV // R − {{}}
    from this obtain x where X = R "" {x} and X ≠ {} by (auto elim: quotientE)
    from this have x ∈ A using ⟨equiv A R⟩ equiv-class-eq-iff by fastforce
    from this ⟨X = R "" {x}⟩ show X ∈ A // R by (auto intro!: quotientI)
  qed
next
  show A // R ⊆ UNIV // R − {{}}
  proof
    fix X
    assume X ∈ A // R
    from this have X ≠ {} using ⟨equiv A R⟩ in-quotient-imp-non-empty by auto
    moreover from ⟨X ∈ A // R⟩ have X ∈ UNIV // R by (metis UNIV-I assms proj-Eps proj-preserves)
    ultimately show X ∈ UNIV // R − {{}} by simp
  qed
qed

1.1.2 Dedicated Facts for Bijection Proof

lemma equiv-relation-of-partition-of:
  assumes equiv A R
  shows \{(x, y). \exists X\in A // R. x ∈ X ∧ y ∈ X\} = R
proof
show \{(x, y). \exists X \in A // R. x \in X \land y \in X\} \subseteq R
proof
fix xy
assume xy \in \{(x, y). \exists X \in A // R. x \in X \land y \in X\}
from this obtain x y where xy = (x, y)
and X \in A // R and x \in X y \in X
by auto
from \{X \in A // R\} obtain z where X = R "\{z\}
by (auto elim: quotientE)
show xy \in R
using (xy = (x, y)): X = R "\{z\}\ <x \in X \ <y \in X\ <equiv A R:\by simp
have R "\{x\} \in A // R
using (equiv A R): (x, y) \in R.
by (simp add: equiv-class-eq-iff quotientI)
moreover have x \in R "\{x\}
using (x, y) \in R: (equiv A R)
by (meson equiv-class-eq-iff equiv-class-self)
moreover have y \in R "\{y\}
using (x, y) \in R: (equiv A R) by simp
ultimately have (x, y) \in \{(x, y). \exists X \in A // R. x \in X \land y \in X\}
by auto
from this show xy \in \{(x, y). \exists X \in A // R. x \in X \land y \in X\}
by simp
qed
qed

1.1.3 Bijection Proof

lemma bij-betw-partition-of:
bij-betw (\lambda R. A // R) \{R. equiv A R\} \{P. partition-on A P\}
proof (rule bij-betw-byWitness[where \(f' = \lambda P. \{(x, y). \exists X \in P. x \in X \land y \in X\}\])
show \forall R \in \{R. equiv A R\}. \{(x, y). \exists X \in A // R. x \in X \land y \in X\} = R
by (simp add: equiv-relation-of-partition-of)
show \forall P \in \{P. partition-on A P\}. A // \{(x, y). \exists X \in P. x \in X \land y \in X\} = P
by (simp add: partition-on-eq-quotient)
show \(\lambda R. A // R\) \cdot \{R. equiv A R\} \subseteq \{P. partition-on A P\}
using partition-on-quotient by auto
show \(\lambda P. \{(x, y). \exists X \in P. x \in X \land y \in X\\} \cdot \{P. partition-on A P\} \subseteq \{R.
equiv A R
  using equiv-partition-on by auto
qed

lemma bij-betw-partition-of-equiv-with-k-classes:
  bij-betw (\lambda R. A // R) \{ R. equiv A R \land card (A // R) = k \} { P. partition-on A P \land card P = k } 
proof (rule bij-betw-byWitness[where f'="\lambda P. \{ (x, y). \exists X \in P. x \in X \land y \in X \}"])
  show \forall R \in \{ R. equiv A R \land card (A // R) = k \}. \{ (x, y). \exists X \in A // R. x \in X \land y \in X \} = R
    by (auto simp add: equiv-relation-of-partition-of)
  show \forall P \in \{ P. partition-on A P \land card P = k \}. A // \{ (x, y). \exists X \in P. x \in X \land y \in X \} = P
    by (auto simp add: partition-on-eq-quotient)
  using partition-on-quotient by auto
  show (\lambda P. \{ (x, y). \exists X \in P. x \in X \land y \in X \} \subseteq \{ R. equiv A R \land card (A // R) = k \})
    by (auto simp add: partition-on-quotient)
  show (\lambda P. \{ P. partition-on A P \land card P = k \}) \subseteq \{ R. equiv A R \land card (A // R) = k \}
    by (auto simp add: partition-on-eq-quotient)
qed

1.2 Finiteness of Equivalence Relations

lemma finite-equiv:
  assumes finite A
  shows finite \{ R. equiv A R \}
proof
  have bij-betw (\lambda R. A // R) \{ R. equiv A R \} \{ P. partition-on A P \}
    by (rule bij-betw-partition-of)
  from this show finite \{ R. equiv A R \}
    using finite A \land finitely-many-partition-on by (simp add: bij-betw-finite)
qed

1.3 Cardinality of Equivalence Relations

theorem card-equiv-rel-eq-card-partitions:
  card \{ R. equiv A R \} = card \{ P. partition-on A P \}
proof
  have bij-betw (\lambda R. A // R) \{ R. equiv A R \} \{ P. partition-on A P \}
    by (rule bij-betw-partition-of)
  from this show card \{ R. equiv A R \} = card \{ P. partition-on A P \}
    by (rule bij-betw-same-card)
qed

corollary card-equiv-rel-eq-Bell:
  assumes finite A
  shows card \{ R. equiv A R \} = Bell (card A)
using assms Bell-altdef card-equiv-rel-eq-card-partitions by force
corollary card-equiv-rel-eq-sum-Stirling:
  assumes finite A
  shows card { R. equiv A R } = sum (Stirling (card A)) {..card A}
using assms card-equiv-rel-eq-Bell Bell-Stirling-eq by simp

theorem card-equiv-k-classes-eq-card-partitions-k-parts:
  card { R. equiv A R ∧ card (A // R) = k } = card { P. partition-on A P ∧ card P = k }
proof
  have bij-betw (λR. A // R) { R. equiv A R ∧ card (A // R) = k } { P. partition-on A P ∧ card P = k }
    by (rule bij-betw-partition-of-equiv-with-k-classes)
  from this show card { R. equiv A R ∧ card (A // R) = k } = card { P. partition-on A P ∧ card P = k }
    by (rule bij-betw-same-card)
qed

corollary
  assumes finite A
  shows card { R. equiv A R ∧ card (A // R) = k } = Stirling (card A) k
using card-equiv-k-classes-eq-card-partitions-k-parts card-partition-on[OF ⟨finite A⟩] by metis
end

2 Cardinality of Partial Equivalence Relations

theory Card-Partial-Equiv-Relations
imports Card-Equiv-Relations
begin

2.1 Definition of Partial Equivalence Relation

definition partial-equiv :: 'a set ⇒ ('a × 'a) set ⇒ bool
where
  partial-equiv A R = (R ⊆ A × A ∧ sym R ∧ trans R)

lemma partial-equivI:
  assumes R ⊆ A × A sym R trans R
  shows partial-equiv A R
using assms unfolding partial-equiv-def by auto

lemma partial-equiv-iff:
  shows partial-equiv A R ⟷ (∃ A' ⊆ A. equiv A' R)
proof
  assume partial-equiv A R
  from ⟨partial-equiv A R⟩ have R "" A ⊆ A
    unfolding partial-equiv-def by blast
moreover have equiv \((R \sim A) R\)
proof (rule equivI)
  from (partial-equiv A R; show sym R
    unfolding partial-equiv-def by blast
  from (partial-equiv A R; show trans R
    unfolding partial-equiv-def by blast
show refl-on \((R \sim A) R\)
proof (rule refl-onI)
  show \(R \subseteq R \sim A \times R \sim A\)
  proof
    fix \(p\)
    assume \(p \in R\)
    obtain \(x y\) where \(p = (x, y)\) by fastforce
    moreover have \(x \in R \sim A\)
      using \(p \in R; p = (x, y); \) (partial-equiv A R);
      partial-equiv-def sym-def by fastforce
    moreover have \(y \in R \sim A\)
      using \(p \in R; p = (x, y); \) \(R \sim A \subseteq A\); \(x \in R \sim A\) by blast
    ultimately show \(p \in R \sim A \times R \sim A\) by auto
  qed
next
  fix \(y\)
  assume \(y \in R \sim A\)
  from this obtain \(x\) where \((x, y) \in R\) by auto
  from \((x, y) \in R\) have \((y, x) \in R\)
    using \(\text{sym } R\) by (meson symE)
  from \((x, y) \in R; (y, x) \in R\) show \((y, y) \in R\)
    using \(\text{trans } R\) by (meson transE)
  qed
qed
ultimately show \(\exists A' \subseteq A. \text{equiv } A' R\) by blast
next
  assume \(\exists A' \subseteq A. \text{equiv } A' R\)
  from this obtain \(A'\) where \(A' \subseteq A\) and \(\text{equiv } A' R\) by blast
  show partial-equiv A R
proof (rule partial-equivI)
  from \(\text{equiv } A' R; A' \subseteq A; \) show \(R \subseteq A \times A\)
    using equiv-class-eq-iff by fastforce
  from \(\text{equiv } A' R; \) show \(\text{sym } R\)
    using equivE by blast
  from \(\text{equiv } A' R; \) show \(\text{trans } R\)
    using equivE by blast
  qed
qed

2.2 Construction of all Partial Equivalence Relations for a
Given Set

definition all-partial-equivs-on :: \(\forall a \set \Rightarrow (\{a \times a\} \set) \set\)
where

\[
\text{all-partial-equivs-on } A = \\
do \\
\k \leftarrow \{0..\text{card } A\}; \\
A' \leftarrow \{A'. A' \subseteq A \land \text{card } A' = k\}; \\
\{R. \text{equiv } A' R\} \\
\}
\]

**lemma** \text{all-partial-equivs-on}:

- **assumes** finite \( A \)
- **shows** \( \text{all-partial-equivs-on } A = \{R. \text{partial-equiv } A R\} \)

**proof**

- **show** \( \text{all-partial-equivs-on } A \subseteq \{R. \text{partial-equiv } A R\} \)
  - **fix** \( R \)
  - **assume** \( R \in \text{all-partial-equivs-on } A \)
  - **from this obtain** \( A' \text{ where } A' \subseteq A \text{ and equiv } A' R \)
    - **unfolding** \text{all-partial-equivs-on-def} **by** auto
  - **from this have** \( \text{partial-equiv } A R \)
    - **using** partial-equiv-iff **by** blast
  - **from this show** \( R \in \{R. \text{partial-equiv } A R\} \)
  - **qed**

- **next**
  - **show** \( \{R. \text{partial-equiv } A R\} \subseteq \text{all-partial-equivs-on } A \)
    - **fix** \( R \)
    - **assume** \( R \in \{R. \text{partial-equiv } A R\} \)
    - **from this obtain** \( A' \text{ where } A' \subseteq A \text{ and equiv } A' R \)
      - **using** partial-equiv-iff **by** (metis mem-Collect-eq)
    - **moreover have** \( \text{card } A' \in \{0..\text{card } A\} \)
      - **using** (\( A' \subseteq A \) finite \( A \)) **by** (simp add: card-mono)
    - **ultimately show** \( R \in \text{all-partial-equivs-on } A \)
      - **unfolding** \text{all-partial-equivs-on-def}
        - **by** (auto simp del: atLeastAtMost-iff)
  - **qed**

**2.3 Injectivity of the Set Construction**

**lemma** equiv-inject:

- **assumes** equiv \( A R \) equiv \( B R \)
- **shows** \( A = B \)

**proof**

- **from assms have** \( R \subseteq A \times A \) \( R \subseteq B \times B \) **by** (simp add: equiv-type)+
- **moreover from assms have** \( \forall x \in A. (x, x) \in R \forall x \in B. (x, x) \in R \)
  - **by** (simp add: eq-eq-equiv-class)+
- **ultimately show** \( \text{thesis} \)
  - **using** mem-Sigma-iff subset-antisym subset-eq **by** blast
  - **qed**

7
lemma injectivity:
assumes \((A' \subseteq A \land \text{card } A' = k) \land (A'' \subseteq A \land \text{card } A'' = k')\)
assumes \(\equiv A' R \land \equiv A'' R'\)
assumes \(R = R'\)
shows \(A' = A'' \land k = k'\)
proof –
  from \(\langle R = R' \rangle\) asms \((2)\) show \(A' = A''\)
  using equiv-inject by fast
  from this asms \((1)\) show \(k = k'\) by simp
qed

2.4 Cardinality Theorem of Partial Equivalence Relations

theorem card-partial-equiv:
assumes \(\text{finite } A\)
shows \(\text{card } \{ R. \text{partial-equiv } A R \} = \text{Bell } (\text{card } A + 1)\)
proof –
  let \(?expr = \text{do}\{\}
     k \leftarrow \{0..\text{card } A\};
     A' \leftarrow \{A'. A' \subseteq A \land \text{card } A' = k\};
     \{R. \equiv A' R\}\}
  have \(\text{card } \{ R. \text{partial-equiv } A R \} = \text{card } (\text{all-partial-equivs-on } A)\)
    using \(\langle \text{finite } A \rangle\) by (simp add: all-partial-equivs-on)
  also have \(\text{card } (\text{all-partial-equivs-on } A) = \text{card } ?expr\)
    unfolding all-partial-equivs-on-def ..
  also have \(\text{card } ?expr = (\sum k = 0..\text{card } A. (\text{card } A \text{ choose } k) \ast \text{Bell } k)\)
    proof –
      let \(?S \gg \gg = \text{?comp = ?expr}\{\}
        fix \(k\)
        assume \(k: k \in \{0..\text{card } A\}\)
        let \(?expr = \text{?comp k}\)
        let \(?S \gg \gg = \text{?comp = ?expr}\)
        have \(\text{finite } ?S\) using \(\langle \text{finite } A \rangle\) by simp
        moreover {\}
          fix \(A'\)
          assume \(A': A' \in \{A'. A' \subseteq A \land \text{card } A' = k\}\)
          from this have \(A' \subseteq A\) and \(\text{card } A' = k\) by auto
          let \(?expr = \text{?comp A'}\)
          have \(\text{finite } A'\)
            using \(\langle \text{finite } A \rangle\) \(\langle A' \subseteq A \rangle\) by (simp add: finite-subset)
          have \(\text{card } ?expr = \text{Bell } k\)
            using \(\langle \text{finite } A \rangle\) \(\langle \text{finite } A' \rangle\) \(\langle A' \subseteq A \rangle\) \(\langle \text{card } A' = k \rangle\)
            by (simp add: card-equiv-rel-eq-Bell)
          moreover have \(\text{finite } ?expr\)
            using \(\langle \text{finite } A' \rangle\) by (simp add: finite-equiv)
          ultimately have \(\text{finite } ?expr\) \(\land\) \(\text{card } ?expr = \text{Bell } k\) by blast
moreover have disjoint-family-on ?comp ?S
  by (injectivity-solver rule: injectivity(1))
moreover have card ?S = card A choose k
  using (finite A: by (simp add: n-subsets))
ultimately have card ?expr = (card A choose k) * Bell k (is - = ?formula)
  by (subst card-bind-constant) auto
moreover have finite ?expr
  using (finite ?S) inner by (auto intro: finite-bind)
ultimately have finite ?expr ∧ card ?expr = ?formula by blast
}
moreover have finite ?S by simp
moreover have disjoint-family-on ?comp ?S
  by (injectivity-solver rule: injectivity(2))
ultimately show card ?expr = (∑ k = 0..card A. (card A choose k) * Bell k)
  by (subst card-bind) auto
qed
also have . . . = (∑ k≤card A. (card A choose k) * Bell k)
  by (auto intro: sum.cong)
also have . . . = Bell (card A + 1)
  using Bell-recursive-eq by simp
finally show ?thesis .
qed

References


